A Scalable Approach to Capacitated Arc Routing Problems Based on Hierarchical Decomposition

Tang, Ke; Wang, Juan; Li, Xiaodong; Yao, Xin

DOI:
10.1109/TCYB.2016.2590558

Document Version
Peer reviewed version

Citation for published version (Harvard):
Tang, K, Wang, J, Li, X & Yao, X 2016, 'A Scalable Approach to Capacitated Arc Routing Problems Based on Hierarchical Decomposition', *IEEE Transactions on Cybernetics*, vol. PP, no. 99, pp. 1-13.
https://doi.org/10.1109/TCYB.2016.2590558

A Scalable Approach to Capacitated Arc Routing Problems based on Hierarchical Decomposition

Ke Tang, *Senior Member, IEEE,* Juan Wang, Xiaodong Li, *Senior Member, IEEE,* and Xin Yao, *Fellow, IEEE*

***Abstract*—The Capacitated Arc Routing Problem (CARP) is a challenging optimization problem with lots of applications in the real world. Numerous approaches have been proposed to tackle this problem. Most of these methods, albeit showing good performance on CARP instances of small and median sizes, do not scale well to large-scale CARPs, e.g., taking at least a few hours to achieve a satisfactory solution on a CARP instance with thousands of tasks. In this paper, an efficient and scalable approach is proposed for CARPs. The key idea of the proposed approach is to hierarchically decompose the tasks involved in a CARP instance into sub-groups and solve the induced sub-problems recursively. The output of the sub-problems at the lower layer in the hierarchy is treated as virtual tasks and new sub-problems are formulated based on these virtual tasks using clustering techniques. By this means, the number of tasks (or virtual tasks) decreases rapidly from the bottom to the top layers of the hierarchy, and the sizes of all sub-problems at each layer can be kept tractable even for very large-scale CARPs. Empirical studies are conducted on CARP instances with up to 3584 tasks, which are an order of magnitude larger than the number of tasks involved in all CARP instances investigated in the literature. The results show that the proposed approach significantly outperforms existing methods in terms of scalability. Since the proposed hierarchical decomposition scheme is designed to obtain a good permutation of tasks in a CARP instance, it may also be generalized to other hard optimization problems that can be formulated as permutation-based optimization problems.**

Index *Terms*—Scalability, Hierarchical Decomposition, Capacitated Arc Routing Problem, Combinatorial Optimization, Clustering

I. Introduction

The Capacitated Arc Routing Problem (CARP) [1] is a classical combinatorial optimization problem that seeks an optimal set of routes to cover a certain subset of edges and/or arcs in a given network subject to some specific constraints, where each edge typically stands for a road in the real world [2, 3]. For its wide range of practical applications, such as winter gritting [4], urban waste collection [5, 6] and snow removal [7, 8], CARP has drawn considerable attentions in the past few decades and a large number of algorithms have been proposed [2, 7, 9-16]. However, previous investigations are mostly limited to relatively small scale CARPs. The largest CARP instance that has been used in the literature, the EGL-G benchmark set [17], consists of up to 375 edges and 375 tasks. In contrast, with the ever growing of big cities, a real-world CARP might involve much more roads and tasks. For example, the central area of Beijing, China, consists of more than 3000 main roads. Hence, it is natural to ask whether existing approaches can still tackle such large-scale CARP instances satisfactorily.

In spite of its importance, the scalability issue of CARP solvers has been rarely addressed in the literature. Prior to 2008, most algorithms for CARP were only tested on small and medium-scale CARP instances, e.g., the *gdb* [18], *val* [19], and Beullens' benchmark sets [14], for most of which the optimal solutions can be found by exact methods. The above-mentioned EGL-G instances were proposed in 2008 and widely used as an additional test set since then. Results obtained on these instances show that the performance of existing approaches clearly deteriorates with the increasing size of CARP instance, both in terms of solution quality (no optimal solution can be found for any EGL-G instance) and in terms of computational cost (less than 10 seconds for a small-scale *val* instance but about 30 minutes for an EGL-G instance) [2, 20, 21].

Motivated by the above observation, Mei et al. [22-24] proposed several approaches to tackle large-scale CARPs. These methods share a similar iterative search framework called Cooperative Co-evolution (CC) [25-28]. That is, a CARP instance is decomposed into a set of sub-problems through dividing its tasks into groups. The sub-problems are tackled separately. The obtained partial-solutions are combined into a complete solution to the original CARP instance and evaluated. The best-so-far complete solution is used to reset the decomposition in the next iteration. In these approaches, decomposition (i.e., grouping tasks) is conducted either randomly [22] or based on a predefined route distance matrix [23, 24], and different optimization techniques can be adopted to solve the sub-problems. These CC-based approaches, e.g., the Route Distance Grouping scheme with Memetic Algorithm with Extended Neighborhood Search (RDG-MAENS) [23], perform significantly better than previous approaches on EGL-G instances. Such advantages should mainly be credited to solving the problem in a divide-and-conquer manner. Nevertheless, these CC-based methods all decompose CARP in

K. Tang and J. Wang are with the USTC-Birmingham Joint Research Institute in Intelligent Computation and Its Applications, School of Computer Science and Technology, University of Science and Technology of China, Hefei, Anhui, China (e-mail: ketang@ustc.edu.cn; jingze@mail.ustc.edu.cn).

X. Li is with the Evolutionary Computation and Machine Learning Research Group, School of Computer Science and Information Technology, RMIT University, Melbourne VIC 3000, Australia (e-mail: xiaodong.li@rmit.edu.au).

X. Yao is with the Center of Excellence for Research in Computational Intelligence and Applications, School of Computer Science, University of Birmingham, Birmingham B15 2TT, U.K. (email: x.yao@cs.bham.ac.uk).

a linear way. That is, to keep the sub-problems at a tractable size, the number of sub-problems needs to increase linearly with the number of tasks. In this case, the complexity of appropriate task groupings increases rapidly with the size of CARP. Consequently, it is more likely that inappropriate groupings of tasks will be obtained on large-scale CARPs and mis-guide the search. Although Shang et al. proposed to improve the CC-based RDG-MAENS [23] in [29], the improved RDG-MAENS was not tested on the existing largest CARP instances (i.e., the *egl-large* instances with up to 375 tasks). Thus, the CC-based methods may still encounter scalability issues on large-scale CARPs.

This paper aims to develop a novel approach that can scale well to large-scale CARPs. Specifically, we are interested in CARPs that are at least an order of magnitude larger than the existing benchmark instances, and aim to develop methods for achieving good solutions to such CARPs within acceptable time, e.g., less than half an hour. A Scalable Approach based on Hierarchical Decomposition (SAHiD) is proposed for this purpose. The key idea of SAHiD is to hierarchically (rather than linearly) decompose the tasks involved in a CARP instance into sub-groups. At the bottom layer of the hierarchy, tasks are decomposed into a few groups and a sub-problem is solved for each group. At each intermediate layer of the hierarchy, the output of the sub-problems at the lower layer is treated as virtual tasks and new sub-problems are formulated based on these virtual tasks rather than the original tasks. The final solution is obtained at the top layer of the hierarchy. With such a hierarchical structure, the number of tasks (or virtual tasks) exponentially decreases from the bottom to the top layers of the hierarchy. Thus, the number of task groups required at each layer, except for the bottom layer, is significantly less than that required for linear decomposition schemes. As a result, the complexity of grouping tasks can be better controlled and inappropriate groupings are less likely to be obtained. Furthermore, as will be shown by our empirical studies, a solution to large-scale CARPs can be obtained efficiently in such a hierarchical way, thus allowing repeating the process in an iterative manner to achieve better solution quality than existing methods in shorter runtime.

The rest of this paper is organized as follows. First, the problem definition and notations of CARP are introduced in Section II. After that, the hierarchical decomposition scheme and detailed steps of SAHiD are described in Sections III and IV, respectively. Empirical studies are presented in Section V to assess the performance of SAHiD and compare it against state-of-the-art CARP solvers. Finally, Section VI concludes the paper and discusses directions for future research.

II. Problem Definition and Notations

An undirected/directed CARP is defined on a connected and undirected/directed graph $G(V,E)$, where V and E represent the sets of vertices and edges respectively. A cost $c(e) > 0$ and a demand $d(e) \geq 0$ are associated with each edge $e \in E$. The edges with positive demands constitute the task set T, i.e., $T = \{\tau \in E | d(\tau) > 0\}$. A vertex $v_0 \in V$ is predefined as the depot, in which a fleet of vehicles are located. The aim of CARP is to determine a set of routes for the vehicles to serve all the tasks with minimal total costs, subject to the following constraints:

1) Each route must start and end at the depot;
2) Each task is served exactly once (but the corresponding edge can be traversed more than once);
3) The total demand of tasks served in each route cannot exceed the vehicle capacity Q.

A solution of CARP can be represented by a sequence of vertices, which directly indicates the order of vertices for the vehicles to visit. However, given a sequence of the tasks, the minimum cost can be easily achieved by summing up the shortest paths between the vertices of each pair of consecutive tasks in the sequence in polynomial time [15]. Since the task representation is more compact, it is adopted in this work. That is a solution to CARP is represented by $s = (R_1, 0, R_2 \dots, R_m)$, where m is the number of routes, the k th route $R_k = (\tau_{k,1}, \dots, \tau_{k,l_k})$, and 0 stands for a dummy task that separates two routes. For each R_k, $\tau_{k,t}$ and l_k denote the tth task and the number of tasks served in route R_k, respectively. More formally, let $u_1(\tau)$ and $u_2(\tau)$ represent the endpoints of task τ, and $inv(\tau)$ the inverse direction of τ, i.e., $u_1\big(inv(\tau)\big) = u_2(\tau)$ and $u_2\big(inv(\tau)\big) = u_1(\tau)$, a CARP can be formulated as follows:

$$\text{minimize } TC(s) \tag{1}$$

s.t.:

$$\sum_{k=1}^{m} l_k = |T| \tag{2}$$

$$\tau_{k_1,i_1} \neq \tau_{k_2,i_2},\ \forall (k_1,i_1) \neq (k_2,i_2) \tag{3}$$

$$\tau_{k_1,i_1} \neq inv(\tau_{k_2,i_2}),\ \forall (k_1,i_1) \neq (k_2,i_2) \tag{4}$$

$$\sum_{i=1}^{l_k} d\left(\tau_{k,i}\right) \leq Q,\ \forall k = 1,2,\dots,m; \tag{5}$$

The objective function, i.e., Eq. (1), requires minimizing the total cost $TC(s)$:

$$TC(s) = \sum_{k=1}^{m} RC(R_k) \tag{6}$$

$RC(R_k)$ is the total cost of route R_k and can be computed using Eq. (7)

$$RC(R_k) = \sum_{i=1}^{l_k} c\big(\tau_{k,i}\big) + dc\left(v_0, u_1\big(\tau_{k,1}\big)\right) + \sum_{i=2}^{l_k} dc\big(u_2(\tau_{k,i-1}), u_1(\tau_{k,i})\big) + dc(u_2(\tau_{k,l_k}), v_0) \tag{7}$$

where $dc(v_i, v_j) > 0$ stands for the deadheading cost induced by traversing the shortest path from vertices v_i to v_j.

In constraints (3) and (4), the inequality $(k_1, i_1) \neq (k_2, i_2)$ means that the two equalities $k_1 = k_2$ and $i_1 = i_2$ do not hold simultaneously. These two constraints prohibit that a task to be served more than once, either in the same route or different routes. Thus, constraints (2) to (4) ensure that all the tasks are served exactly once. Constraint (5) indicates that the total demand of each route should not exceed the vehicle capacity Q.

The challenge of CARP can be viewed from two perspectives. First, the optimal permutation of tasks corresponds to the optimal solution of CARP, and the latter can

be obtained from the former in polynomial time [15]. Second, if the optimal assignments of tasks to vehicles are available, the optimal solution of CARP can also be obtained by solving several independent single-vehicle problems that are considerably easier than CARP. This paper takes the first perspective, i.e., the proposed approach aims to identify a good permutation efficiently.

III. HIERARCHICAL DECOMPOSITION OF CARP

It can be observed that the scale of a CARP instance mainly depends on the number of tasks to be served. Furthermore, a CARP can be addressed in two steps, i.e., finding a permutation of the task set and dividing this permutation into feasible routes. The optimal permutation must correspond to an optimal feasible solution, and vice versa [15, 30]. More importantly, given a permutation of tasks, the corresponding best feasible solution can be acquired in polynomial time [15]. Therefore, the key challenge to CARP can be viewed as finding the optimal permutation of tasks. The proposed Hierarchical Decomposition (HD) scheme is essentially a method for finding a good permutation of tasks efficiently. To be concrete, HD introduces a number of *virtual tasks* to construct a hierarchical structure, as demonstrated in Fig. 1. In the figure, each node (in the lth layer) of the hierarchy corresponds to a virtual task τ_i^l, which is a permutation of several tasks. K_{l-1} represents the number of virtual tasks included in layer l. It will be discussed in detail in Section A. Each node at the bottom layer (i.e., layer 1) corresponds to a real task. The node at the top layer (i.e., layer h) represents a permutation of all the tasks. The hierarchy is built in a bottom-up way. At the bottom layer (i.e., layer 1), tasks are grouped and ordered within each group. The permutation of tasks in each group is treated as a virtual task at layer 2. This procedure is executed on the obtained virtual tasks recursively until only 1 virtual task remains, which is a permutation of all the tasks. For example, suppose there are 4 tasks $\{\tau_1^1, \tau_2^1, \tau_3^1, \tau_4^1\}$ at layer 1, each 2 of them are connected to a node at layer 2, the virtual task corresponding to this node, denoted by τ_i^2 is a permutation of the 2 tasks, e.g., $\tau_1^2 = (\tau_2^1, \tau_1^1)$ and $\tau_2^2 = (\tau_3^1, \tau_4^1)$. τ_1^2 and τ_2^2 are then grouped and ordered, forming a virtual task at layer 3, e.g., $\tau_1^3 = (\tau_2^2, \tau_1^2) = (\tau_3^1, \tau_4^1, \tau_2^1, \tau_1^1)$. By this means, a permutation of the 4 tasks is obtained.

From the above description, for a given CARP instance, the HD scheme starts from the bottom layer and recursively group tasks into virtual tasks of a larger size until only 1 virtual task remains. Each node at non-bottom layers requires solving a sub-problem to find the optimal permutation of the tasks (virtual tasks) assigned to that node, which is a partial-solution to the problem of finding optimal permutation of all tasks. Algorithm 1 demonstrates the general framework of the HD scheme. The benefits offered by such a hierarchy are two-folds. From the perspective of problem decomposition, a linear decomposition scheme (e.g., as adopted by the CC-based approaches) has to involve a large number of small-size sub-problems to cope with large-scale CARPs. Meanwhile, task grouping itself is a non-trivial problem, the complexity of which increases exponentially with the number of task groups (i.e., number of sub-problems). Hence, it is highly likely that an inappropriate grouping will be obtained on large-scale CARPs, or a significant computational cost is needed to identify a good grouping of tasks. In other words, the performance of linear decomposition will deteriorate rapidly with the problem-size of CARP. As a result, the solution quality may also deteriorate rapidly since the grouping of tasks significantly affects the search course. In contrast, the HD scheme allows the number of nodes (i.e., virtual tasks) to decrease exponentially from the bottom layer towards the top layer. Thus, the cost for identifying a suitable grouping of tasks increases slowly with the scale of CARP. Hence, the scalability (in terms of the quality of task grouping) of HD is expected to be better than that of a linear grouping, and can lead to a better grouping of tasks (and thereby solution quality) than linear decomposition, especially when the total time budget for solving a CARP is limited and task grouping needs to be done as fast as possible. From the perspective of search effectiveness, since a sub-problem (node) of an upper layer only takes the partial-solutions obtained at the lower layers as its input, but does not change the inner structure of the partial-solutions, the hierarchical structure naturally allows searching at different step-sizes by solving sub-problems at different layers and may lead to a more effective search.

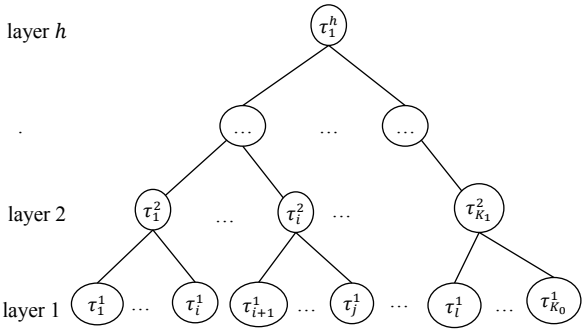

Fig. 1. The hierarchical structure of HD

The HD strategy involves 2 design issues. That is, how to group virtual tasks (i.e., line 2-3 in Algorithm 1) and how to find the optimal permutation in a group of virtual tasks (i.e., line 4 in Algorithm 1). These will be detailed below.

A. Grouping Virtual Tasks

Intuitively, tasks close to each other are more likely to be served successively in solutions with high quality. Hence, a natural idea is to assign neighboring tasks to the same group. Thus the closeness between two virtual tasks needs to be defined in order to group the virtual tasks. An intuitive idea is to take the deadheading cost between two virtual tasks as the closeness. Since the deadheading cost is defined between two vertices rather than two virtual tasks (paths) and 4 different deadheading costs can be obtained by connecting different pairs of endpoints of two virtual tasks, the closeness between virtual task τ_i^l and τ_j^l is defined as the average deadheading cost between them:

$$\Delta(\tau_i^l, \tau_j^l) = \frac{dc(v_i,v_j)+dc(u_i,u_j)+dc(v_i,u_j)+dc(u_i,v_j)}{4} \quad (8)$$

Procedure HD(VT)
Input: virtual task set VT
Output: a permutation of tasks PT
1: **repeat**
2: randomly choose the cluster number $K \in [1, \beta \cdot |VT|]$;
3: divide VT into groups by using k-means;
4: order the virtual tasks within each group;
5: $VT \leftarrow$ {permutation of tasks in each group};
6: **until** $|VT| = 1$
7: **return** the permutation of tasks in VT;

Algorithm 1 The Hierarchical Decomposition HD(VT)

where (v_i, u_i) and (v_j, u_j) denote the start and end vertices of τ_i^l and τ_j^l, respectively. $dc(v_i, v_j)$ is the deadheading cost between vertices v_i and v_j. As a special case, the closeness between virtual task τ_i^l and the depot v_0 is $\Delta(v_0, \tau_i^l) = \frac{dc(v_i,v_0)+dc(u_i,v_0)}{2}$.

Given the closeness measure of two virtual tasks, grouping virtual tasks can be formulated as a clustering problem. In principle, any clustering method can be applied for this purpose. We choose the well-known k-means algorithm [31] for its simplicity. The k-means algorithm requires calculating the centroid of each cluster. This can be easily done in a real-space but cannot be directly computed in case of clustering virtual tasks. Hence, in each iteration of k-means, the centroid of a cluster is defined as the virtual task with minimal average closeness to other virtual tasks in the same cluster.

The number of clusters, K, is a user-defined parameter in k-means and has a great influence on the clustering results. In the HD scheme, since the clustering process is invoked at each layer and the number of virtual tasks varies over layers, a fixed value of K for all clustering processes is inappropriate. Note that each virtual task at layer l corresponds to a cluster obtained at layer $l-1$. Hence, let K_l denote the number of clusters obtained at layer l, we set it as an integer randomly generated within $[1, \beta \cdot K_{l-1}]$, where $\beta \in (0, 1)$ is a pre-defined parameter. For the bottom layer (i.e., layer 1), $K_1 = |T|$.

Another important issue related to the effectiveness of k-means is the selection of initial cluster centroids. The initial centroids can be selected randomly or by using specific methods [32-34]. Here we use a simple heuristic that disperse the centroids as widely as possible. Specifically, the heuristic works by adding non-centroid virtual tasks into the centroid set one by one such that the sum of the closeness between the newly added virtual task and existing ones in the set is maximal. It should be noted that the depot is regarded as a dummy virtual task and added to the centroid set at the beginning.

B. Ordering virtual tasks

At each node of the hierarchical structure, the permutation of a subset of tasks or virtual tasks needs to be optimized. Recall that the capacity constraints are not considered when solving these sub-problems, since the HD scheme aims to achieve the optimal permutation of all tasks regardless of capacity constraints. There exist lots of heuristic methods for finding such a permutation-based problem. Since a number of such sub-problems need to be solved for building the hierarchy, it is not worth adopting a time-consuming method. For this consideration, we employ a greedy search heuristic named Best Insertion Heuristic (BIH). BIH firstly chooses the nearest virtual task to the depot in terms of deadheading cost. Then, at each iteration, the virtual task with the minimal deadheading cost to the current endpoint of the path is added to the end of the path. If multiple virtual tasks satisfy this condition, only one is randomly chosen. The process terminates when all virtual tasks have been added to the path.

Procedure HDU(VT)
Input: virtual task set VT
Output: a feasible solution s;
1: apply HD(VT) to generate a permutation of tasks PT;
2: apply Ulusoy's splitting procedure to partition PT into a solution s;
3: **return** s;

Algorithm 2 The Hierarchical Decomposition and Ulusoy's splitting HDU(VT)

C. Generating a solution to CARP based on hierarchical decomposition

As mentioned before, the HD scheme seeks a good permutation of all tasks in CARP regardless of capacity constraint. Given a permutation of tasks, a solution (with respect to this permutation) to CARP can be obtained by splitting the permutation into a number of routes that satisfy capacity constraints. This can be done with well-established methods in polynomial time [15]. Herein we employ the well-known Ulusoy's splitting procedure [10], an exact method that has been proved to be capable of solving the problem optimally. Thus, the combination of the HD scheme and the Ulusoy's splitting procedure, namely HDU as demonstrated in Algorithm 2, forms our approach to CARPs.

IV. The Scalable Approach based on Hierarchical Decomposition

The HDU described in the previous section obtains a solution to a CARP in a constructive way. It can also be embedded into an iterative search process, which allows the solution obtained using HDU to be further improved. The proposed SAHiD is developed following this idea. Briefly speaking, SAHiD is an individual-based iterative search method. At each iteration, it firstly employs HDU to obtain a solution to the CARP, and then some traditional local search operator is applied to further improve the solution obtained. Specifically, SAHiD involves three phases, i.e., initialization, reconstruction, and local search, as detailed below.

Initialization: At the first iteration, an initial solution, say s, is obtained by applying HDU to the CARP instance. Then, the local search operator is applied to s for further improvement.

Reconstruction: In the reconstruction phase, HDU is applied to generate new candidate solutions. But different from in the *initialization* phase, HDU is not applied to achieve a solution from scratch, i.e., based on the un-ordered set of tasks. Instead, the solution obtained in the last iteration, say s, is first

```
Procedure LS(s)
Input: solution s
Output: potentially improved solution s
repeat
  for each sub-routes SR of each route R in s do
    reverse SR to obtain a new solution s';
    if s' is better than s then
      s ← s';
      break;
    end if
  end for
  until s remains unchanged
if s is not updated then
  apply MS operator to improve s;
  if s is updated then
    repeat
      for each sub-routes SR of each route R in s do
        reverse SR to obtain a new solution s';
        if s' is better than s then
          s ← s';
          break;
        end if
      end for
      until s remains unchanged
    end if
end if
return s;
```

Algorithm 3 The local search procedure LS(s)

randomly split into a number of virtual tasks. Specifically, each route of s is split into two virtual tasks with a predefined probability α, resulting in a set of virtual tasks. Then, HDU is applied to this set to obtain a new solution. Since s is built based on the original set of tasks and polished with local search, it is expected to contain some useful pattern of a good solution. Splitting the routes does not change the permutation of tasks in the same virtual task, and thus are likely to keep the useful patterns. Hence, by applying HDU to the virtual tasks rather than the original tasks, the useful pattern in the previous solution can be exploited, and thus benefit the search for a better solution.

Local search: When a solution is obtained by HDU, either in the initialization or reconstruction phase, a first improvement local search procedure is applied to further improve it. The local search starts with a reverse move operator similar to the 2-opt operator [15] for a single route, i.e., it reverses the direction of a sub-route (i.e. part of a route). Suppose the reverse operator is applied to a route consisting of t tasks. At each iteration, all possible sub-routes are enumerated with the length of sub-route increasing from 1 to $t-1$. During this course, the current solution is updated once a solution with a lower cost is found. This procedure terminates when all sub-routes of each route is checked at least once, and the whole local search procedure terminates if the solution obtained by HDU is updated at least once.

If the reverse operator fails to improve the solution obtained by HDU, the Merge-Split (MS) operator [16] is applied to conduct a best improvement search. That is, at each step, all solutions that can be reached by the MS operator from the current solution are examined and the best and improved one is chosen to replace the current solution. Compared to the reverse operator, MS is a search operator with a larger step-size, and thus is more likely to escape from the current solution, which is a local optimum. Interested readers are referred to [16] for detailed steps of MS. If the MS operator manages to find a better solution, the reverse operator will be applied to the improved solution again to exploit the new local region. Otherwise, the whole local search procedure terminates with the solution obtained by HDU remaining unchanged.

```
Procedure SAHiD(T)
Input: task set T
Output: a feasible solution s*
1:  generate an initial solution s using HDU(T);
2:  apply LS(s) to improve s;
3:  s* ← s
4:  while stopping criteria are not met do
5:    generate a virtual task set VT by splitting the routes of s;
6:    generate a solution s' using HDU(VT);
7:    apply LS(s') to improve s';
8:    if s' is acceptable then
9:      s ← s'
10:     if s' is better than s* then
11:       s* ← s'
12:     end if
13:   end if
14: end while
15: return s*;
```

Algorithm 4 The pseudo code of SAHiD(T)

The pseudo-code of the local search procedure is presented in Algorithm 3. Note that HDU always produces feasible solutions, the reverse operator only changes the order of tasks within a feasible route, and the MS operator also always generates feasible routes. Hence, no infeasible solution will be produced during the search process of SAHiD. For this reason, only the total costs are taken into account when comparing two solutions.

Algorithm 4 depicts the steps of SAHiD. It is noteworthy that the best solution found so far is stored in an external archive (line 10-12) and outputted as the final solution. It might be inappropriate to keep the best solution in the search process of SAHiD if it cannot be improved for a long time. Otherwise, the search will be stuck at this local best solution. Hence, we adopt the Threshold Accepting idea [35] in SAHiD. Given a solution s, if no better solution is found after σ consecutive iterations, a new solution worse than s will still be accepted (i.e., replace s) as long as its quality is not worse than θ% of that of the best-found solution. Finally, the SAHiD can be terminated either when a predefined time budget is used up or no better solution is found for a predefined number of iterations.

V. Experimental Studies

To evaluate the effectiveness of SAHiD, two sets of empirical studies have been conducted to compare SAHiD against a number of state-of-the-art approaches to CARPs. In the first study, the performance of different algorithms is examined in terms of the time required to achieve a predefined solution quality. In the second study, the algorithms are compared from the perspective of solution quality obtained using a predefined time budget. In addition, further empirical analysis has also been conducted to assess the contribution of the HD scheme to SAHiD.

TABLE I
THE PARAMETER SETTINGS OF HCOLS

Name	Description	Value
β	Scale parameter in HD (Sub-Section III.A)	0.1
α	Probability of partitioning a route (Section IV)	0.1
p	Parameter of the MS operator (Section IV)	2
θ	Parameter of Threshold accepting (Section IV)	110%
σ	Maximum number of idle iterations for accepting an ascending move (Section IV)	10000

TABLE II
THE INFORMATION OF INSTANCES IN *Hefei* AND *Beijing* TEST SET

Name	$\|V\|$	$\|E\|$	$\|T\|$	Proportion of tasks	Q	M
Hefei-1	850	1212	121	10%	9000	7
Hefei-2	850	1212	242	20%	9000	14
Hefei-3	850	1212	364	30%	9000	19
Hefei-4	850	1212	485	40%	9000	28
Hefei-5	850	1212	606	50%	9000	35
Hefei-6	850	1212	727	60%	9000	42
Hefei-7	850	1212	848	70%	9000	49
Hefei-8	850	1212	970	80%	9000	56
Hefei-9	850	1212	1091	90%	9000	63
Hefei-10	850	1212	1212	100%	9000	69
Beijing-1	2820	3584	358	10%	25000	7
Beijing-2	2820	3584	717	20%	25000	11
Beijing-3	2820	3584	1075	30%	25000	18
Beijing-4	2820	3584	1433	40%	25000	23
Beijing-5	2820	3584	1792	50%	25000	30
Beijing-6	2820	3584	2151	60%	25000	36
Beijing-7	2820	3584	2509	70%	25000	41
Beijing-8	2820	3584	2868	80%	25000	47
Beijing-9	2820	3584	3226	90%	25000	52
Beijing-10	2820	3584	3584	100%	25000	58

A. Benchmark Set

Since this work mainly studies the scalability of search methods, we are more interested in the performance on large-scale CARPs rather than small scale ones. Furthermore, as mentioned before, most of the existing small or median-scale CARP benchmark instances can be solved near optimally in a rather short time period (e.g., 10 seconds). Therefore, instead of using the more traditional benchmark sets, two new sets of CARP instances, namely *Hefei* and *Beijing* test sets[1], are generated. The *Hefei* set is generated from the map of the Hefei city in China, which consists of 1212 main roads (i.e., edges). The *Beijing* set is generated from the central area (the area inside the 5th ring road) of Beijing, China, which consists of 3584 main roads. For each set, 10 instances are generated by randomly setting part of the edges as tasks. The number of tasks for each set increases from 10% to 100% of the number of edges with a step-size 10%. TABLE II shows the detailed information of these two benchmark sets. It can be observed that these two sets involves instances that are one order of magnitude larger than the largest CARP benchmark instances used in the literature, i.e., EGL-G, which consists of at most 375 edges and 375 tasks. Furthermore, since the major challenge of CARP depends on the number of tasks rather than the number of edges, the *Hefei* and *Beijing* sets allow assessing the scalability of an algorithm in this regard. It should also be noted that the performance of an algorithm on a CARP is also affected by the capacity constraints. Thus the same capacity constraints are set to all instances in the same set, so as to focus our investigation on scalability. M is the minimal number of vehicles required to serve all the tasks, which is obtained as follows:

$$M = \left\lceil \frac{\Sigma_{\tau \in T} d(\tau)}{Q} \right\rceil \quad (9)$$

B. Compared algorithms

Three algorithms, including Variable Neighborhood Search (VNS) [7], Tabu Search Algorithm 1 (TSA1) [17] and RDG-MAENS [23], are chosen for our comparative studies. VNS and TSA1 are both individual-based search approaches for CARPs. They have shown appealing performance not only in terms of solution quality, but also (and more importantly in the context of this work) in terms of efficiency. RDG-MAENS is an approach dedicated to large-scale CARPs and has been shown to outperform other population-based search methods, e.g., MAENS [16], on large-scale CARP instances. Thus, it was chosen as the state-of-the-art representative population-based search methods for CARPs.

C. Experimental Protocol

To make a fair comparison, all algorithms involved in the empirical studies are implemented in C++ and run on the same computing platform, i.e., Intel® Core™ i7-4790 processor with 3.6 GHz. For all experiments presented hereafter, the results are obtained by executing the algorithms for 25 independent runs. Each of the tested algorithms consists of a few parameters to be predefined.

SAHiD has 5 user-defined parameters, i.e., the scale parameter β in HD; the probability α of partitioning a route in the reconstruction phase; parameter p introduced by the MS operator [16] and θ and σ introduced by threshold accepting strategy [35, 36]. Since the last three parameters are proposed in previous works for local search, but are not introduced by the HD decomposition scheme, the values suggested in the original publications are directly adopted. For the parameters α and β, a sensitivity analysis is carried out to test the performance of SAHiD as well as choosing parameter values for more comprehensive empirical studies. Specifically, five values (0.1, 0.3, 0.5, 0.7 and 0.9) were tested for both parameters, which led to 25 combinations of parameters values. The sensitivity analysis is conducted on four typical instances, including *Hefei 1*, *Hefei 10*, *Beijing 1* and *Beijing10*. For each instance, SAHiD is executed for 10 times with each of the 25 parameter combinations (i.e., 250 runs in total). Fig. 2 depicts the heat maps of the average results (over 10 runs) obtained by SAHiD with different values of α and β. It can be observed that α is more critical to performance on larger instances (*Hefei 10* and *Beijing 10*), while β mainly affects the performance of SAHiD on smaller instances. The reason is that, HD is the core of SAHiD, the tree size obtained by HD has a great influence on the algorithm performance. Both α and β affect the size of tree, α determines the number of leaf nodes and β controls the upper bound of number of intermediate nodes. As described in

[1] Instances of the two test sets is available at http://staff.ustc.edu.cn/~ketang/codes/LSCARPset.zip

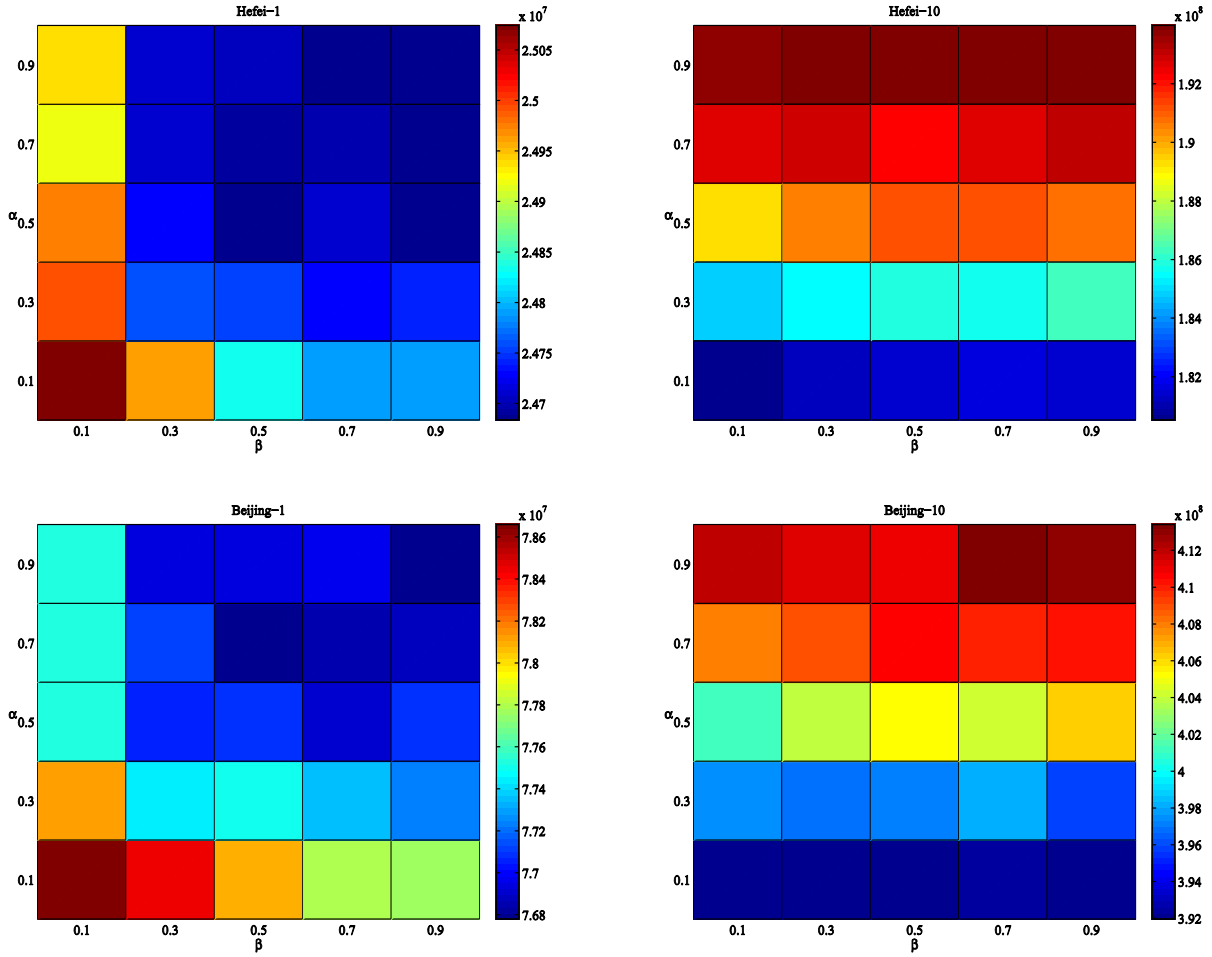

Fig. 2. Heat map of average total cost obtained by SAHiD with different values of α and β on instances *Hefei-1,Hefei-10,Beijing-1* and *Beijing-1*0. Blue indicates better results and red stands for worse results.

Section III.A, the number of nodes in layer l is within $[1,\ \beta \cdot K_{l-1}]$, where K_{l-1} is the number of nodes in layer $l-1$. Hence, the number of intermediate nodes is not only affected by β but also determined by the number of bottom nodes (leaf nodes), which means α plays a more important role than β when the problem size is large. On the smaller instances (*Hefei 1* and *Beijing 1*), the number of routes is relatively small. Thus, a larger α is needed to ensure a sufficient number of leaf nodes (virtual tasks), if there are very few leaf nodes, the role of the hierarchy will be greatly reduced. For the larger instances (i.e., *Hefei 10* and *Beijing 10*), on the other hand, a small value of α already led to splitting a large number of routes. In fact, TABLE I indicates that the routes (i.e.， the number of vehicles, M) in a solution to *Hefei 10* is about 10 times larger than that in a solution to *Hefei 1*. Thus, setting α to 0.9 and 0.1 for *Hefei 1* and *Hefei 10* actually resulted in comparable number of routes being split. Hence, as a rule of thumb, α is suggested to take the value that will lead to the split of around 7 routes in the reconstruction phase. Since β does not appear to affect SAHiD as much as α, 0.1 can be used as the default value. The results reported in this paper were all obtained with $\alpha = 0.1$ and $\beta = 0.1$.

TABLE I summarizes the parameter settings for SAHiD. For the compared VNS [7], TSA1 [17] and RDG-MAENS [23], the best parameter settings reported in the original publications are employed. By this means, we hope to keep the comparison as fair as possible.

D. Comparison in terms of runtime

Runtime is one of the most important issues when investigating the scalability of an algorithm. Ideally, the runtime for an algorithm to achieve the optimal solution or a solution within a given approximation ratio should be tested. However, such an analysis cannot be done for large-scale CARP instances used in this work, because the optimal solution and the lower bound on the total cost are unknown for the instances. Hence, we resort to a threshold of total costs as the target for the compared algorithms. Specifically, for the *Hefei* and *Beijing* sets, SAHiD is firstly run for 25 times. A time budget of 30 seconds is given for each run on each instance of *Hefei* and *Beijing* sets. For each instance, the total costs of the 25 final solutions are recorded. The largest one among these 25

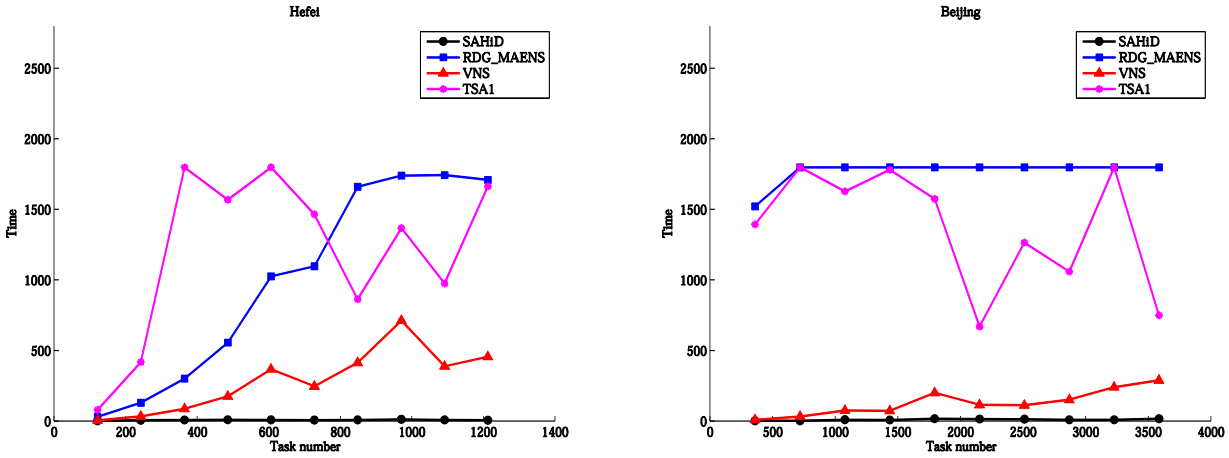

Fig. 3. Average computational time (to achieve a predefined solution quality) versus the number of tasks over all the instances of *Hefei* and *Beijing* sets for each compared algorithm.

values indicates the worst performance of SAHiD in the 25 runs. Thus, it is used as the target for other compared algorithms. The runtime for each of the other algorithms to achieve the same solution quality for the first time is recorded. Since no matter how the target solution quality is chosen, it is possible that an algorithm may not reach this target or takes extremely long time to reach it, the compared algorithms are terminated if a larger time budget of 30 minutes is used up.

Fig. 3 depicts the average runtime for the compared algorithms on the *Hefei* and *Beijing* sets. Each point in the figure corresponds to the runtime of an algorithm on an instance. Note that, since the worst total costs obtained by SAHiD are used as the target, the average runtime required by SAHiD on each instance is always not greater than 30 seconds for *Hefei* and *Beijing* sets. From the figure, it can be observed that VNS, TSA1 and RDG-MAENS all consume much more runtime than SAHiD. In fact, some even fail to reach the target solution quality in 30 minutes for some large-scale instances. For example, RDG-MAENS fails in all 25 runs on all the *Beijing* instances except for *Beijing-1*, on which the average runtime consumed is still close to the given time budget. TSA1 also reaches the given time budget on 2 out of the 20 instances, and always takes more than 500 seconds to reach the target solution quality. VNS appears to be more efficient than RDG-MAENS and TSA1, but the runtime consumed by it show a growth trend with respect to the problem size and thus its scalability is not as good as SAHiD. These observations clearly demonstrate the superiority of SAHiD to the compared algorithms.

E. Comparison in terms of solution quality

In addition to runtime, another important characteristic of an algorithm is the solution quality that can be achieved with a given time budget. This is evaluated by our second experiment. Specifically, each algorithm is given 30 minutes to search for the solution to a CARP instance in *Hefei* and *Beijing* benchmark sets. Further, since the EGL-G benchmark set [17] has been used to evaluate TSA1 and RDG-MAENS in the original publications, this set of instances is also employed in the experiment. The time budget is set to 15 minutes as the scale of EGL-G is much smaller than *Hefei* and *Beijing* sets.

TABLEs III to V present the costs of the final solutions obtained by the compared algorithms on the three test sets. The first 5 columns present the basic information of instances. For each compared algorithm, the columns headed "Best" and "Average" provide the best and average costs among the 25 runs, respectively. The last column headed "Std" present the standard deviations calculated over the 25 runs. The minimal average results are marked with "*". On each instance, SAHiD is compared to RDG-MAENS, VNS and TSA1 by using Wilcoxon rank-sum test with the level of significance 0.05 over 25 runs, results highlighted in bold/underline indicate that the corresponding algorithm is significantly better/worse than SAHiD on the corresponding instance. Results without any symbol indicate that the difference between SAHiD and the corresponding algorithm is statistically insignificant.

From TABLEs III and IV, it can be observed that SAHiD significantly outperforms the other algorithms in terms of solution quality. It achieves the smallest total costs on 7 out of 10 instances in *Hefei* and 9 out of 10 instances in *Beijing*. Statistical tests also confirmed that the significant difference between SAHiD and the other algorithms. The results in TABLE V are more mixed. To be specific, SAHiD is not as competitive as RDG-MAENS on the 10 EGL-G instances. In comparison to TSA1, SAHiD performs better on 6 instances, i.e., EGL-G1-D, EGL-G1-E, EGL-G2-B, EGL-G2-C, EGL-G2-D and EGL-G2-E, while is inferior on the other 4 instances. SAHiD still outperforms VNS on all 10 instances. The inferior performance of SAHiD to RDG-MAENS on the EGL-G set is understandable, because the former is designed with the aim to tackle large-scale CARPs. As the size of EGL-G is relatively small, a more costly but more precise search method like RDG-MAENS should be able to find a better solution while the computational time is still acceptable. In addition, unlike the *Hefei* and *Beijing* sets, different EGL-G instances are subject to different capacity constraints. Thus,

TABLE III
RESULTS ON *Hefei* BENCHMARK SET IN TERMS OF THE TOTAL SOLUTION COSTS. "BEST" AND "AVERAGE" STAND FOR THE BEST AND AVERAGE RESULTS OBTAINED FROM 25 INDEPENDENT RUNS. "STD" STANDS FOR THE STANDARD DEVIATION. THE MINIMAL AVERAGE RESULTS ARE MARKED WITH "*". FOR EACH INSTANCE, BOLD (UNDERLINED) RESULTS INDICATE THAT THE CORRESPONDING ALGORITHM IS BETTER (WORSE) THAN SAHiD BASED ON WILCOXON RANK-SUM TEST WITH THE LEVEL OF SIGNIFICANCE 0.05. "# OF 'W-D-L'" SUMMARIZES THE NUMBER OF 'WIN-DRAW-LOSE' OF SAHiD VERSUS THE OTHER ALGORITHMS.

Name	$\|V\|$	$\|T\|$	$\|E\|$	Q	SAHiD			RDG-MAENS			VNS			TSA1		
					Best	Average	Std	Best	Average	Std	Best	Average	Std	Best	Average	Std
Hefei-1	850	1212	121	9000	248048	251024	1820	246221	**247341***	2293	245596	**247819**	2745	250155	252615	1591
Hefei-2	850	1212	242	9000	441574	445376	2476	436020	**441539***	4142	436637	449979	5375	447853	456228	5539
Hefei-3	850	1212	364	9000	586880	590969	2305	583050	**589152***	2697	588682	595263	3108	623795	637201	8003
Hefei-4	850	1212	485	9000	754015	759402*	2495	754855	761351	4362	763256	774323	6394	774182	791790	5481
Hefei-5	850	1212	606	9000	964772	976276*	4742	980153	991813	5755	984121	994794	6109	1019224	1042701	11496
Hefei-6	850	1212	727	9000	1095530	1106735*	5318	1119584	1132063	8966	1110030	1128667	9404	1134041	1162641	13806
Hefei-7	850	1212	848	9000	1299430	1309474*	4792	1329745	1361125	14356	1322290	1337353	6745	1339160	1353502	6235
Hefei-8	850	1212	970	9000	1474390	1483694*	4857	1526453	1550509	13695	1492790	1517151	12477	1521857	1537169	6709
Hefei-9	850	1212	1091	9000	1648840	1659700*	6103	1705381	1749079	18872	1675790	1694957	10164	1696706	1716256	9236
Hefei-10	850	1212	1212	9000	1793890	1808860*	7836	1837767	1923264	31697	1834860	1852622	10183	1873504	1901167	12679
# of "w-d-l"									6-1-3			9-0-1			10-0-0	

TABLE IV
RESULTS ON *Beijing* BENCHMARK SET IN TERMS OF THE TOTAL SOLUTION COSTS. "BEST" AND "AVERAGE" STAND FOR THE BEST AND AVERAGE RESULTS OBTAINED FROM 25 INDEPENDENT RUNS. "STD" STANDS FOR THE STANDARD DEVIATION. THE MINIMAL AVERAGE RESULTS ARE MARKED WITH "*". FOR EACH INSTANCE, BOLD (UNDERLINED) RESULTS INDICATE THAT THE CORRESPONDING ALGORITHM IS BETTER (WORSE) THAN SAHiD BASED ON WILCOXON RANK-SUM TEST WITH THE LEVEL OF SIGNIFICANCE 0.05. "# OF 'W-D-L'" SUMMARIZES THE NUMBER OF 'WIN-DRAW-LOSE' OF SAHiD VERSUS THE OTHER ALGORITHMS.

Name	$\|V\|$	$\|T\|$	$\|E\|$	Q	SAHiD			RDG-MAENS			VNS			TSA1		
					Best	Average	Std	Best	Average	Std	Best	Average	Std	Best	Average	Std
Beijing-1	2820	3584	358	25000	775523	784727	5591	812647	829406	12688	774502	782415*	4452	813907	829132	6340
Beijing-2	2820	3584	717	25000	1167480	1183955*	8431	1303570	1337954	18939	1168190	1192292	10196	1353567	1401363	25378
Beijing-3	2820	3584	1075	25000	1586180	1605846*	9231	1777852	1847922	33258	1591540	1618484	11888	1678224	1709279	14801
Beijing-4	2820	3584	1434	25000	1910880	1936994*	11694	2126151	2193399	34159	1920330	1953892	16746	2053938	2070885	14532
Beijing-5	2820	3584	1792	25000	2273080	2298630*	16879	2581910	2639458	32481	2293120	2335915	23040	2396483	2440319	26726
Beijing-6	2820	3584	2151	25000	2664510	2707500*	18433	2968102	3047295	41112	2705060	2743677	18024	2774161	2814735	22018
Beijing-7	2820	3584	2509	25000	3013590	3038157*	15658	3331900	3388263	26081	3015790	3063813	25226	3147294	3186240	22426
Beijing-8	2820	3584	2868	25000	3283530	3313590*	21925	3584696	3697025	44951	3323850	3366215	24686	3415275	3456037	22381
Beijing-9	2820	3584	3226	25000	3621490	3684250*	32404	3934270	4061793	49504	3653630	3723830	45148	3890129	3943883	37089
Beijing-10	2820	3584	3584	25000	3935540	4004310*	29488	4206005	4353966	51063	4002040	4040694	27384	4066188	4103532	15501
# of "w-d-l"									10-0-0			9-1-0			10-0-0	

another interesting observation from TABLE V is that SAHiD outperforms TSA1 mainly in the cases where the capacity constraints are tighter. A possible reason might be that SAHiD always visits feasible solutions during the search, while TSA1 may generate infeasible solutions and thus only part of the time budget is used for searching in the feasible region.

Furthermore, the performance of the algorithm is also tested using different time budgets less than the above given time budgets. The average costs obtained over 25 runs are plotted in Fig. 4. For the sake of brevity, only the results on 2 instances are provided for each benchmark set. The results on other instances follow a similar pattern and are made available in the online appendix[2]. The figure further confirms the superiority of SAHiD on large scale instances. For instance, the curve for SAHiD is always beneath the curves of the other algorithms on *Hefei-10* and *Beijing-10*, indicating that SAHiD can always perform the best among the 4 algorithms if a solution is needed with a tighter time budget on a large scale instance. As for the EGL-G instances, it can be found that curves of SAHiD drop rapidly with time, but level off later. RDG-MAENS improves the solution slower than SAHiD at the beginning, but continues to improve it. This observation is consistent with the expectation that RDG-MAENS is a more costly but more precise method, which might achieve better solutions than SAHiD if the problem size is moderate in comparison to the time budget.

F. Analysis of the contribution of HD to SAHiD

Since the core component of SAHiD is the HD scheme. It is also interesting to investigate whether the HD scheme is indispensable for SAHiD. For this purpose, another algorithm namely SArandom is developed. The only difference between SAHiD and SArandom is that the latter does not use the HD scheme in the reconstruction phase (line 6 in Algorithm 4).

[2] Available at http://staff.ustc.edu.cn/~ketang/codes/SAHiDresults.pdf

TABLE V
RESULTS ON EGL-G BENCHMARK SET IN TERMS OF THE TOTAL SOLUTION COSTS. "BEST" AND "AVERAGE" STAND FOR THE BEST AND AVERAGE RESULTS OBTAINED FROM 25 INDEPENDENT RUNS. "STD" STANDS FOR THE STANDARD DEVIATION. THE MINIMAL AVERAGE RESULTS ARE MARKED WITH "*". FOR EACH INSTANCE, BOLD (UNDERLINED) RESULTS INDICATE THAT THE CORRESPONDING ALGORITHM IS BETTER (WORSE) THAN SAHID BASED ON WILCOXON RANK-SUM TEST WITH THE LEVEL OF SIGNIFICANCE 0.05. "# OF 'W-D-L'" SUMMARIZES THE NUMBER OF 'WIN-DRAW-LOSE' OF SAHID VERSUS THE OTHER ALGORITHMS.

Name	$\|V\|$	$\|T\|$	$\|E\|$	Q	SAHiD			RDG-MAENS			VNS			TSA1		
					Best	Average	Std	Best	Average	Std	Best	Average	Std	Best	Average	Std
EGL-G1-A	255	375	347	28600	1017030	1033874	6588	1002347	**1014177***	5898	1048220	1061032	8238	1017089	1034531	9933
EGL-G1-B	255	375	347	22800	1139860	1155899	7812	1124610	**1133625***	5960	1160620	1191821	13372	1130096	**1145438**	10493
EGL-G1-C	255	375	347	19000	1270110	1281702	7131	1252061	**1261541***	6933	1303100	1324484	12496	1263835	1287478	12070
EGL-G1-D	255	375	347	16200	1397490	1422693	7387	1386044	**1397790***	6851	1430320	1465704	15134	1411361	1433744	14015
EGL-G1-E	255	375	347	14100	1549540	1565029	6856	1527473	**1542491***	9543	1575240	1614683	12059	1563234	1597127	16095
EGL-G2-A	255	375	375	28000	1128040	1141925	8592	1108242	**1122013***	7280	1142490	1166879	14840	1119328	**1130786**	9423
EGL-G2-B	255	375	375	23100	1238740	1253578	7236	1221077	**1233611***	6088	1279280	1293706	8402	1242060	1258284	7418
EGL-G2-C	255	375	375	19400	1374000	1386629	7823	1355956	**1369175***	7305	1411810	1439277	13400	1378920	1398971	14892
EGL-G2-D	255	375	375	16700	1505570	1529137	9698	1485341	**1506033***	8679	1563960	1587437	14493	1522390	1555457	15661
EGL-G2-E	255	375	375	14700	1656860	1673222	7754	1637063	**1649882***	7608	1694730	1724093	13871	1672677	1704553	21171
# of 'w-d-l'									0-0-10			10-0-0			6-2-2	

TABLE VI
RESULTS OF SAHID AND SARANDOM ON ALL TEST SETS IN TERMS OF THE TOTAL SOLUTION COSTS. "BEST" AND "AVERAGE" STAND FOR THE BEST AND AVERAGE RESULTS OBTAINED FROM 25 INDEPENDENT RUNS. "STD" STANDS FOR THE STANDARD DEVIATION. FOR EACH INSTANCE, BOLD (UNDERLINED) RESULTS INDICATE THAT SARANDOM IS BETTER (WORSE) THAN SAHID BASED ON WILCOXON RANK-SUM TEST WITH THE LEVEL OF SIGNIFICANCE 0.05.

Name	SAHiD			SArandom-			RPD(%)
	Best	Average	Std	Best	Average	Std	
EGL-G1-A	1017030	1033874	6588	1038660	1051855	6843	1.74
EGL-G1-B	1139860	1155899	7812	1159420	1182700	10447	2.32
EGL-G1-C	1270110	1281702	7131	1298250	1312731	7861	2.42
EGL-G1-D	1397490	1422693	7387	1444100	1461256	7968	2.71
EGL-G1-E	1549540	1565029	6856	1594180	1613778	7966	3.11
EGL-G2-A	1128040	1141925	8592	1145130	1163482	10036	1.89
EGL-G2-B	1238740	1253578	7236	1263190	1282771	7966	2.33
EGL-G2-C	1374000	1386629	7823	1414250	1433035	9519	3.35
EGL-G2-D	1505570	1529137	9698	1562830	1577365	9240	3.15
EGL-G2-E	1656860	1673222	7754	1715300	1730814	8614	3.44
							2.65(mean)
Hefei-1	248048	251024	1820	246265	**247401**	776	-1.44
Hefei-2	441574	445376	2476	439437	445329	3580	-0.01
Hefei-3	586880	590969	2305	589521	598517	3321	1.28
Hefei-4	754015	759402	2495	778721	786735	4942	3.6
Hefei-5	964772	976276	4742	1005620	1024557	8398	4.95
Hefei-6	1095530	1106735	5318	1161690	1179338	8389	6.56
Hefei-7	1299430	1309474	4792	1376010	1399276	14850	6.86
Hefei-8	1474390	1483694	4857	1578120	1603242	11507	8.06
Hefei-9	1648840	1659700	6103	1777520	1808153	12278	8.94
Hefei-10	1793890	1808860	7836	1953270	1986672	17833	9.83
							4.86(mean)
Beijing-1	776379	784891	6223	767512	**777961**	5902	-0.86
Beijing-2	1180010	1188836	6589	1170610	1182402	7381	-0.13
Beijing-3	1593720	1610383	8198	1632690	1658375	16942	3.27
Beijing-4	1927850	1943053	9055	1985840	2036163	28357	5.12
Beijing-5	2273080	2298630	16879	2439860	2504124	38455	8.94
Beijing-6	2664510	2707500	18433	2885540	3004417	86641	10.97
Beijing-7	3013590	3038157	15658	3281750	3382513	77413	11.33
Beijing-8	3283530	3313590	21925	3564160	3728470	103502	12.52
Beijing-9	3621490	3684250	32404	3972310	4162260	128888	12.97
Beijing-10	3935540	4004310	29488	4292190	4515296	114710	12.76
							7.69(mean)

Instead, SArandom randomly merges virtual tasks into permutations. As in Sub-Section V.E, SArandom is applied to the three benchmark sets and the results are summarized in TABLE VI. The relative percentage of deviation (RPD) is given in the last column for each instance. RPD is the ratio of the difference between the average costs obtained by SAHiD and SArandom to the average costs obtained by SArandom. It measures the improvement that can be achieved by replacing

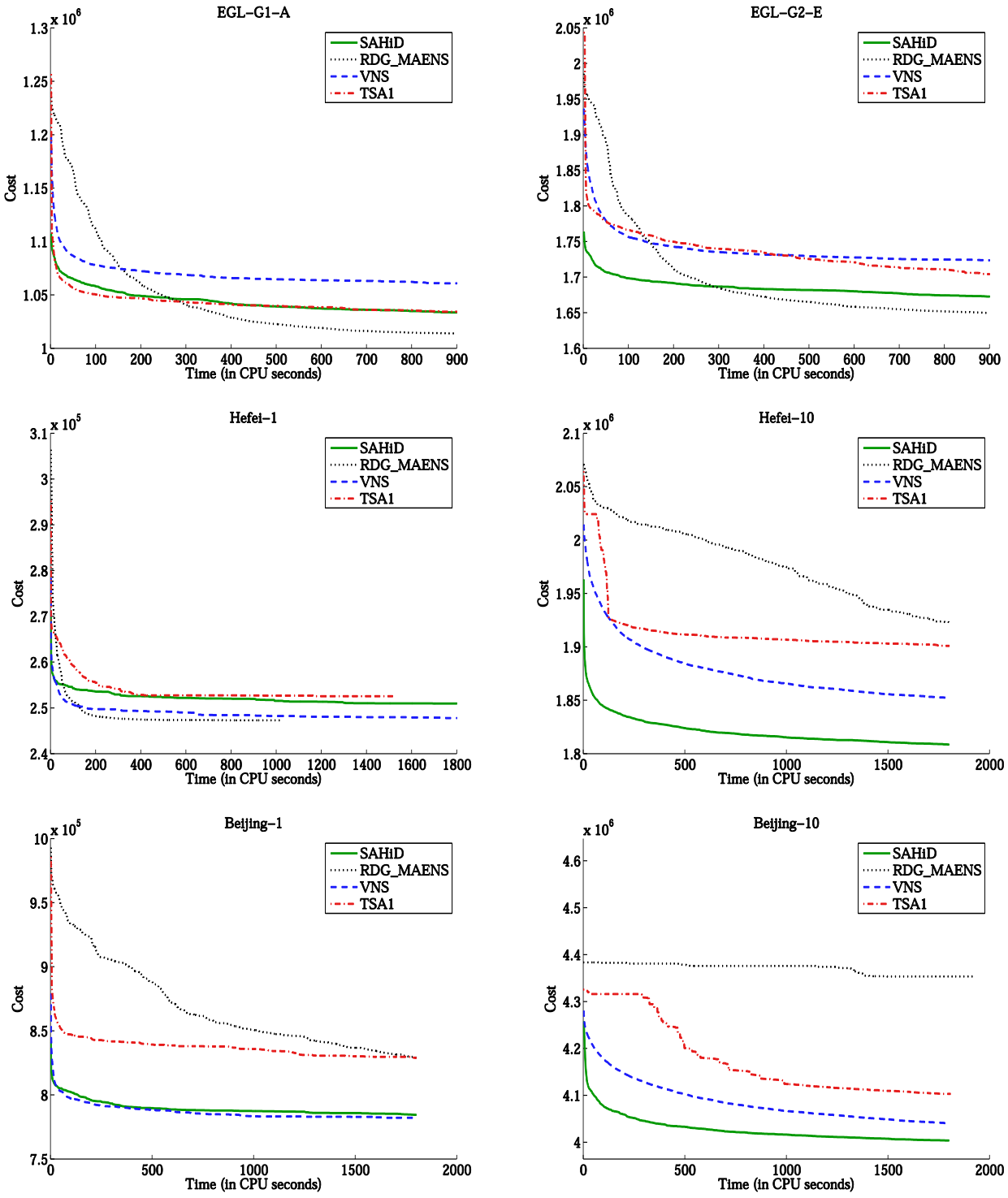

Fig. 4. The convergence curves of SAHID, RDG-MAENS, VNS and TSA1 on instances EGL-G1-A, EGL-G2-E, *Hefei-1,Hefei-10,Beijing-1* and *Beijing-10*.

the random permutation scheme with the HD scheme, i.e., a larger RPD indicates a more significant influence of HD on SAHiD.

TABLE VI shows that SAHiD outperforms SArandom on 28 out of 30 test instances and the gap between them shows a growing trend with respect to the problem size (e.g., the average RPD on EGL-G, *Hefei* and *Beijing* are 2.65%, 4.86% and 7.69%, respectively). Moreover, comparing TABLE VI to Tables III and IV reveals that the results of SArandom are significantly worse than those of RDG-MAENS, VNS and TSA1. Specifically, SArandom achieves worse performance than TSA1 on 11 out of 20 instances in *Hefei* and *Beijing* sets (the largest 5 of *Hefei* instances and 6 of *Beijing* instances). When compared to RDG-MAENS, SArandom performs worse

on all the *Hefei* instances and the largest 3 *Beijing* instances (i.e., *Beijing-8, Beijing-9* and *Beijing-10*). The gap between SArandom and VNS is more obvious, VNS beats SArandom on 16 out of 20 instances in *Hefei* and *Beijing* sets (8 of *Hefei* and 8 of *Beijing*). This fact confirms that the advantages of SAHiD over the other compared algorithms should be credited to the HD scheme.

VI. Conclusion and Future work

This paper presents a novel approach, namely SAHiD, to CARPs. SAHiD distinguishes from previous methods in the sense that it employs a hierarchical decomposition scheme, which is capable of generating a good permutation of tasks, i.e., an intermediate solution to the CARP, in a very efficient way. By employing the proposed hierarchical decomposition scheme in an iterative search process, SAHiD can tackle CARP instances of large scales. Empirical studies on two new CARP benchmark sets that are one order of magnitude larger than the existing ones show that SAHiD significantly outperforms state-of-the-art methods in terms of both computational time and solution quality (given a time budget less than 30 minutes). Hence, SAHiD can better scale up to large-scale CARPs than the compared methods, particularly in cases when a solution needs to be obtained in a few minutes or even seconds.

The promising performance of SAHiD has pointed to several future research directions. First, in addition to the HD scheme, the other components of SAHiD, e.g., the methods of grouping and ordering virtual tasks in HD and the local search procedure, can be improved using alternative techniques in the literature. Since instances generated from real-world maps may not represent all instances that could be synthesized using the mathematical formulation of CARP. The specific clustering method used in HD might fail in some of the latter cases. Hence, the interactions between them and the decomposition scheme can be further investigated, so as to develop novel components that suit SAHiD better. Second, the experimental results reveal that the capacity constraints may also affect the scalability of an algorithm on CARPs. Although this observation seems to be obvious, it has never been systematically studied and it is unclear how the scalability of CARP solvers can be enhanced in this aspect. Finally, the hierarchical decomposition scheme proposed in this work is in essence a method for efficiently obtaining good permutations of tasks. It can be generalized to other permutation-based optimization problems, e.g., vehicle routing [37, 38], scheduling [39-41] and path planning [42, 43], as long as a suitable closeness measure could be designed. Note that, in practice, some of these problems may need to be solved either in real-time [44, 45] or in dynamic environments [46], which means efficiency is even more important for these problems than for CARP. The idea of hierarchical decomposition is expected to benefit the development of scalable approaches in those domains.

Acknowledgment

This work was supported in part by the National Natural Science Foundation of China (Grant 61329302), the Program for New Century Excellent Talents in University (Grant NCET-12-0512), EPSRC (Grant EP/K001523/1), the Royal Society Newton Advanced Fellowship (Ref. No. NA150123) and ARC Discovery grant (No. DP120102205). The work of X. Yao was also supported by a Royal Society Wolfson Research Merit Award.

References

[1] B. L. Golden and R. T. Wong, "Capacitated arc routing problems," *Networks,* vol. 11, pp. 305-315, 1981.

[2] R. W. Eglese, "Routeing winter gritting vehicles," *Discrete Applied Mathematics,* vol. 48, pp. 231-244, 1994.

[3] V. Maniezzo, "Algorithms for large directed CARP instances: urban solid waste collection operational support," Technical Report UBLCS-2004-16, Department of Computer Science, University of Bologna, Italy 2004.

[4] H. Handa, L. Chapman and X. Yao, "Robust route optimization for gritting/salting trucks: A CERCIA experience," *Computational Intelligence Magazine, IEEE,* vol. 1, pp. 6-9, 2006.

[5] P. Lacomme, C. Prins and W. Ramdane-Cherif, "Evolutionary algorithms for periodic arc routing problems," *European Journal of Operational Research,* vol. 165, pp. 535-553, 2005.

[6] F. Chu, N. Labadi and C. Prins, "A scatter search for the periodic capacitated arc routing problem," *European Journal of Operational Research,* vol. 169, pp. 586-605, 2006.

[7] M. Polacek, K. F. Doerner, R. F. Hartl, and V. Maniezzo, "A variable neighborhood search for the capacitated arc routing problem with intermediate facilities," *Journal of Heuristics,* vol. 14, pp. 405-423, 2008.

[8] J. F. Campbell and A. Langevin, "Roadway snow and ice control," in *Arc Routing*: Springer, 2000, pp. 389-418.

[9] W. L. Pearn, "Approximate solutions for the capacitated arc routing problem," *Computers & Operations Research,* vol. 16, pp. 589-600, 1989.

[10] G. Ulusoy, "The fleet size and mix problem for capacitated arc routing," *European Journal of Operational Research,* vol. 22, pp. 329-337, 1985.

[11] R. Hirabayashi, Y. Saruwatari and N. Nishida, "Tour constructive algorithm for the capacitated arc routing problem," *Asia-Pacific Journal of Operational Research,* vol. 9, pp. 155-175, 1992.

[12] B. L. Golden, J. S. Dearmon and E. K. Baker, "Computational experiments with algorithms for a class of routing problems," *Computers & Operations Research,* vol. 10, pp. 47-59, 1983.

[13] H. Longo, M. P. de Aragão and E. Uchoa, "Solving capacitated arc routing problems using a transformation to the CVRP," *Computers & Operations Research,* vol. 33, pp. 1823-1837, 2006.

[14] P. Beullens, L. Muyldermans, D. Cattrysse, and D. Van Oudheusden, "A guided local search heuristic for the capacitated arc routing problem," *European Journal of Operational Research,* vol. 147, pp. 629-643, 2003.

[15] P. Lacomme, C. Prins and W. Ramdane-Cherif, "Competitive memetic algorithms for arc routing problems," *Annals of Operations Research,* vol. 131, pp. 159-185, 2004.

[16] K. Tang, Y. Mei and X. Yao, "Memetic algorithm with extended neighborhood search for capacitated arc routing problems," *Evolutionary Computation, IEEE Transactions on,*

vol. 13, pp. 1151-1166, 2009.
[17] J. Brand O and R. Eglese, "A deterministic tabu search algorithm for the capacitated arc routing problem," *Computers & Operations Research,* vol. 35, pp. 1112-1126, 2008.
[18] J. S. DeArmon, *A comparison of heuristics for the capacitated Chinese postman problem*: Master's thesis, University of Maryland, College Park, MD, USA, 1981.
[19] E. Benavent, V. Campos, A. Corberan, and E. Mota, "The capacitated arc routing problem: lower bounds," *Networks,* vol. 22, pp. 669-690, 1992.
[20] R. W. Eglese and L. Y. Li, "A tabu search based heuristic for arc routing with a capacity constraint and time deadline," in *Meta-Heuristics*: Springer, 1996, pp. 633-649.
[21] L. Y. Li and R. W. Eglese, "An interactive algorithm for vehicle routeing for winter-gritting," *Journal of the Operational Research Society,* pp. 217-228, 1996.
[22] Y. Mei, X. Li and X. Yao, "Decomposing Large-Scale Capacitated Arc Routing Problems using a Random Route Grouping Method," in *Evolutionary Computation (CEC), 2013 IEEE Congress on*, 2013, pp. 1013-1020.
[23] Y. Mei, X. Li and X. Yao, "Cooperative coevolution with route distance grouping for large-scale capacitated arc routing problems," *Evolutionary Computation, IEEE Transactions on,* vol. 18, pp. 435-449, 2014.
[24] Y. Mei, X. Li and X. Yao, "Variable Neighborhood Decomposition for Large Scale Capacitated Arc Routing Problem," in *Evolutionary Computation (CEC), 2014 IEEE Congress on*, 2014, pp. 1313-1320.
[25] Z. Yang, K. Tang and X. Yao, "Large scale evolutionary optimization using cooperative coevolution," *Information Sciences,* vol. 178, pp. 2985-2999, 2008.
[26] M. N. Omidvar, X. Li, Z. Yang, and X. Yao, "Cooperative co-evolution for large scale optimization through more frequent random grouping," in *Evolutionary Computation (CEC), 2010 IEEE Congress on*, 2010, pp. 1-8.
[27] X. Li and X. Yao, "Cooperatively coevolving particle swarms for large scale optimization," *Evolutionary Computation, IEEE Transactions on,* vol. 16, pp. 210-224, 2012.
[28] W. Gao, G. G. Yen and S. Liu, "A dual-population differential evolution with coevolution for constrained optimization," *Cybernetics, IEEE Transactions on,* vol. 45, pp. 1108-1121, 2015.
[29] R. Shang, K. Dai, L. Jiao, and R. Stolkin, "Improved Memetic Algorithm Based on Route Distance Grouping for Multiobjective Large Scale Capacitated Arc Routing Problems.," *Cybernetics, IEEE Transactions on,* pp. 1000-1013, 2015.
[30] P. Lacomme, C. Prins and W. Ramdane-Chérif, "A genetic algorithm for the capacitated arc routing problem and its extensions," *Applications of Evolutionary Computing,* pp. 473-483, 2001.
[31] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, 1967, pp. 281-297.
[32] M. R. Anderberg, "Cluster analysis for applications," *Academic Press, New York,* 1973.
[33] L. Bobrowski and J. C. Bezdek, "c-means clustering with the l1 and l∞ norms," *Systems, Man and Cybernetics, IEEE Transactions on,* vol. 21, pp. 545-554, 1991.
[34] S. S. Khan and A. Ahmad, "Cluster center initialization algorithm for K-means clustering," *Pattern Recognition Letters,* vol. 25, pp. 1293-1302, 2004.
[35] G. Dueck and T. Scheuer, "Threshold accepting: a general purpose optimization algorithm appearing superior to simulated annealing," *Journal of Computational Physics,* vol. 90, pp. 161-175, 1990.
[36] M. Polacek, R. F. Hartl, K. Doerner, and M. Reimann, "A variable neighborhood search for the multi depot vehicle routing problem with time windows," *Journal of Heuristics,* vol. 10, pp. 613-627, 2004.
[37] J. Luo, X. Li, M.-R. Chen, and H. Liu, "A novel hybrid shuffled frog leaping algorithm for vehicle routing problem with time windows," *Information Sciences*, vol. 316, pp. 266–292, 2015.
[38] L. Tan, F. Lin, and H. Wang, "Adaptive comprehensive learning bacterial foraging optimization and its application on vehicle routing problem with time windows," *Neurocomputing*, vol. 151, pp. 1208–1215, 2015.
[39] Z. Xiao and Z. Ming, "A method of workflow scheduling based on colored petri nets," *Data & Knowledge Engineering*, vol. 70, no. 2, pp. 230–247, 2011.
[40] Z. Chen, M. Qiu, Z. Ming, L. T. Yang, and Y. Zhu, "Clustering scheduling for hardware tasks in reconfigurable computing systems," *Journal of Systems Architecture*, vol. 59, no. 10, pp. 1424–1432, 2013.
[41] J. Li, M. Qiu, J.-W. Niu, L. T. Yang, Y. Zhu, and Z. Ming, "Thermal-aware task scheduling in 3d chip multiprocessor with real-time constrained workloads," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 12, no. 2, p. 24, 2013.
[42] Z.Zhu, J.Xiao, J.-Q.Li, F.Wang and Q.Zhang,"Global path planning of wheeled robots using multi-objective memetic algorithms," *Integrated Computer-Aided Engineering*, vol. 22, no. 4, pp. 387–404, 2015.
[43] A. Macwan, J. Vilela, G. Nejat, B. BenhabibA Multirobot Path-Planning Strategy for Autonomous Wilderness Search and Rescue," *Cybernetics, IEEE Transactions on,* vol. 45(9): 1784-1797, Sept. 2015.
[44] H. Luo, J. Fang, and G. Q. Huang, "Real-time scheduling for hybrid flowshop in ubiquitous manufacturing environment," *Computers & Industrial Engineering*, vol. 84, pp. 12–23, 2015.
[45] J. Li, M. Qiu, Z. Ming, G. Quan, X. Qin, and Z. Gu, "Online optimization for scheduling preemptable tasks on iaas cloud systems," *Journal of Parallel and Distributed Computing*, vol. 72, no. 5, pp. 666– 677, 2012.
[46] S. Nguyen, M. Zhang, M. Johnston and K. C. Tan, "Automatic Programming via Iterated Local Search for Dynamic Job Shop Scheduling," *Cybernetics, IEEE Transactions on*, 45(1): 1-14, Jan. 2015.

Ke Tang (S'05-M'07-SM'13) received the B.Eng. degree from Huazhong University of Science and Technology, Wuhan, China, in 2002, and the Ph.D. degree from Nanyang Technological University, Singapore, in 2007, respectively.

Since 2007, he has been with the School of Computer Science and Technology, University of Science and Technology of China, where he is currently a Professor. He has authored/co-authored more than 100 refereed publications. His major research interests include evolutionary computation, machine learning, and their real-world applications.

Dr. Tang is an Associate Editor of the IEEE Transactions on Evolutionary Computation, IEEE Computational Intelligence Magazine and Computational Optimization and Applications (Springer), and served as a member of Editorial Boards for a few other journals. He is a member of the IEEE Computational Intelligence Society (CIS) Evolutionary Computation Technical Committee and the IEEE CIS Emergent Technologies Technical Committee. He is the recipient of the Royal Society Newton Advanced Fellowship.

Juan Wang received her Ph.D. degree from University of Science and Technology of China, Hefei, China, in 2016. She is now an algorithm engineer of a company in Shenzhen, Guangdong, China.

Her research interests include memetic algorithm and other metaheuristics for solving capacitated arc routing problems.

Xiaodong Li (M'03-SM'07) received his B.Sc. degree from Xidian University, Xi'an, China, and Ph.D. degree in information science from University of Otago, Dunedin, New Zealand, respectively. Currently, he is an Associate Professor at the School of Computer Science and Information Technology, RMIT University, Melbourne, Australia. His research interests include evolutionary computation, neural networks, complex systems, multiobjective optimization, and swarm intelligence. He serves as an Associate Editor of the IEEE Transactions on Evolutionary Computation, Swarm Intelligence (Springer), and International Journal of Swarm Intelligence Research. He is a founding member and currently a Vice-chair of IEEE CIS Task Force on Swarm Intelligence, and a Vice-chair of IEEE Task Force on Multi-modal Optimization, a former chair of IEEE CIS Task Force on Large Scale Global Optimization. He was the General

Chair of SEAL'08, a Program Co-Chair AI'09, and a Program Co-Chair for IEEE CEC'2012. He is the recipient of 2013 ACM SIGEVO Impact Award.

Xin Yao is a Chair Professor of Computer Science and the Director of CERCIA (the Centre of Excellence for Research in Computational Intelligence and Applications) at the University of Birmingham, UK.

He is an IEEE Fellow and the President (2014-15) of IEEE Computational Intelligence Society (CIS). His major research interests include evolutionary computation and ensemble learning. He published 200+ refereed international journal papers. His papers won the 2001 IEEE Donald G. Fink Prize Paper Award, 2015 and 2010 IEEE Transactions on Evolutionary Computation Outstanding Paper Awards, 2010 BT Gordon Radley Award for Best Author of Innovation (Finalist), 2011 IEEE Transactions on Neural Networks Outstanding Paper Award, and many other best paper awards. He received the prestigious Royal Society Wolfson Research Merit Award in 2012 and the IEEE CIS Evolutionary Computation Pioneer Award in 2013. He was the Editor-in-Chief (2003-08) of IEEE Transactions on Evolutionary Computation. He has published frequently on the topic of capacitated arc routing problems (CARP) since his first paper in 2006 (H. Handa, L. Chapman and Xin Yao, ``Robust route optimisation for gritting/salting trucks: A CERCIA experience," IEEE Computational Intelligence Magazine, 1(1):6-9, February 2006.).